12-10-2005

Approximate Methods For Solving Flowshop Problems

Pramod Jain

APPROXIMATE METHODS FOR SOLVING FLOWSHOP PROBLEMS

By

Pramod Jain

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulf llment of the Requirements
for the Degree of Master of Science
in Industrial Engineering
in the Department of Industrial Engineering

Mississippi State, Mississippi

December 2005

APPROXIMATE METHODS FOR SOLVING FLOWSHOP PROBLEMS

By

Pramod Jain

Approved:

Burak Eksioglu Assistant Professor Department of Industrial Engineering (Major Professor)	Mingzhou Jin Assistant Professor Department of Industrial Engineering
Sandra D. Eksioglu Assistant Research Professor Department of Industrial Engineering	Stanley F. Bullington Professor of Industrial Engineering and Graduate Coordinator, Department of Industrial Engineering

Kirk H. Schulz
Dean of the College of Engineering

Name: Pramod Jain

Date of Degree: December 9, 2005

Institution: Mississippi State University

Major Field: Industrial Engineering

Major Professor: Dr. Burak Eksioglu

Title of Study: APPROXIMATE METHODS FOR SOLVING FLOWSHOP PROBLEMS

Pages in Study: 49

Candidate for Degree of Master of Science

The f ow shop scheduling problem is a classical combinatorial problem being studied for years. The focus of this research is to study two variants of the f ow shop scheduling problem in order to minimize makespan by scheduling *n* jobs on *m* machines. A solution approach is developed for the modif ed f ow shop problem with due dates and release times. This algorithm is an attempt to contribute to the limited literature for the problem. Another tabu search-based solution approach is developed to solve the classical f ow shop scheduling problem. This meta-heuristic (called *3XTS*) allows an eff cient search of the neighboring solutions leading to a fast solution procedure. Several control parameters affecting the quality of the algorithm are experimentally tested, and certain rules are established for different problem instances. The *3XTS* is compared to another tabu search method (that seems to be a champion) in terms of solution quality and computation time.

DEDICATION

To my mother, Smt. Shakuntla Jain and my father, Late Sh. R.S Jain.

ACKNOWLEDGMENTS

Sincere appreciation is extended to Dr. Burak Eksioglu for serving as my major advisor and for providing much more than just guidance and assistance for the research described in this thesis and my masters program. I feel short of words in describing the appreciation for him as my true mentor.

I am grateful to Dr. Mingzhou Jin for his ethic, encouragement, valuable suggestions and career guidance.

I thank Dr. Sandra Eksioglu for being there always to solve my problems. Her suggestions and guidance was very helpful.

I thank Dr. Allen Greenwood from the department of Industrial Engineering and Mr. Clay Walden from Center for Advanced Vehicular Systems (CAVS) for providing me financial support in the form of Graduate Research Assistantship for pursuing my research and putting me in industry projects. The research projects brought me face to face to industry problems and will be very useful in my career.

I thank Mr. Travis Hill for spending ample time in correcting my errors in my technical writing, programming and in other areas. He has made me realize the value of hard work.

I wish to express my special thanks to my very good friend Mr. Amit Bugde. He was a constant source of support for me and helped me enormously in programming and other

technical difficulties. He has been very caring, understanding and cheered me up in tough times as a true friend.

A special thanks to Mr. Sucharith Vanguri who has taken loads of time out of his busy schedule to help me out with no complaints. He is a person who is there always with a solution to all my problems.

I would also like to thank my lab mates Mr. Prasad Vemuri and Mr. Chandrashekar K. Nagashekar for healthy discussions and arguments on various topics, and providing a helping hand in my research.

A special and grateful thank you is extended to my mother, my best friends Yulia and Gaurav for their thoughtfulness, love, concern, confidence, and support throughout my graduate studies, for without them, this endeavor would not have been possible.

Special thanks also go to the Department of Industrial Engineering and all the professors who have equipped me with knowledge and skills. A word of appreciation is also extended to Mr. Carl Brown, Ms. Peggy Roach and Ms. Bonnie Ladner for their friendliness and help.

Last, but not the least, I want to express my gratitude to all the people who have helped me in one way or the other: Thank you very much!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Scheduling is def ned as an allocation of limited resources to tasks over time [1]. It is a decision-making process with a goal to optimize one or more objectives. These resources and tasks may take many forms. The resources may be machines in a workshop, runways at an airport, crew at a construction site or processing units in a computing environment and the tasks may be operations at a production process, take-offs and landings at an airport, stages in a construction project or executions of a computer program. Each task may have a different priority level on earliest possible starting time and/or a due-date. The objective may also take many forms such as minimizing makespan, minimizing the total completion times, or minimizing the number of tasks completed after the committed due dates.

Relying largely on mathematical methodology, the formal research of scheduling began in the early 1950s. Since then, scheduling theory has become one of the most active areas in operations research with a signif cant number of problems and models studied in literature. The attraction of scheduling theory is not just for its deep mathematical basis but also for its impressive processes and mission-critical computer systems. See Blazewicz at. al [3] for an overview of scheduling in computer and manufacturing systems.

Traditionally, the research on scheduling is motivated by questions arising in production planning and manufacturing. These questions also led to the development of a theory of machine scheduling. In the last two decades, with many new types of manufacturing shops such as flexible manufacturing systems (FMS) and computer integrated manufacturing (CIM) coming into prominence, the research on machine scheduling has undergone a profound transition [18]. Often scheduling problems in modern manufacturing systems are more complex than classical scheduling problems because in most cases additional resources and constraints have to be satisfied. Also simultaneous decisions regarding connections between several limited resources are to be made. For instance, in many automated manufacturing environments, material handling is performed by computer-controlled robots, cranes, or vehicles, rendering the performance of the systems highly dependent on the interaction between the machines and the material-handling devices [8]. A good schedule that synchronizes the activities of machine processing and material handling may increase throughput rate and also reduce work-in-process and production costs. It is reported that there are more than 600 companies in the United States and Japan which develop and/or use advanced algorithms for the scheduling of material-handling devices [17]. Unfortunately, most of the classical scheduling models do not incorporate several of the distinct attributes of state-of-the-art manufacturing systems.

1.1 Preliminaries

The section's purpose is to provide a brief outline of the classical scheduling models. We make every effort to adhere to traditional notation and standard terminology. The scheduling models discussed in this thesis are based on the deterministic off-line machine scheduling paradigm, where all data are assumed to be discrete (i.e., non-negative integers) and known with certainty in advance. For a rigorous and comprehensive discussion on scheduling models and computational complexity of scheduling algorithms, the books by Cormen et. al. [6], Garey and Johnson [9] and Papadimitriou [23] are excellent sources. The term machine scheduling is mentioned as scheduling throughout the thesis.

1.1.1 Scheduling problems

In scheduling, the limited resources consist of one or more machines, and tasks are modeled as jobs that can be executed by the machines. A task (job) f rst becomes available for processing at its ready time, and it must receive amount of processing equal to its processing time. Typically, a problem in scheduling is characterized by the types of machines and jobs in the system, by the constraints imposed , and by a desired optimality principle.

A characteristic of the machine environment is that a machine can handle, at most, one job at a time, and each job can be processed by only one machine at a time. In general, a machine can begin its next job immediately after the current job is completed, and there are no machine breakdowns at any moment of time. For the scheduling problem considered in this thesis, preemption is not allowed during the processing of any operation, which

means that the execution of a job on a machine will proceed without interruption once it starts. A machine scheduling problem is in fact a sequencing problem where a schedule is completely specif ed by the sequence in which jobs are performed.

In what follows, classical scheduling models are described. The focus is on the shop scheduling models that are frequently encountered in manufacturing shop f oors. Several books on scheduling, e.g. Blazewicz et. al [3], Brucker [4], and Pinedo [24] present a variety of scheduling models, such as scheduling in single and parallel machines.

1.1.2 Shop scheduling models

In many manufacturing and production systems, jobs have to be processed by several machines in a given order. This multi-operation situation is often ref ected in the so-called shop scheduling model, where a number of jobs are to be processed in a shop consisting of several machines. Usually, it is assumed that the machines have unlimited buffer space and a job can be stored in the buffer for an unlimited amount of time. If the machines have limited buffer space, then blocking occurs when the buffer is full. In this case, the job at the upstream machine cannot be released into the buffer after completing its processing and has to remain at the upstream machine. This occurence prevents a job in queue at that machine from beginning its processing.

To further def ne shop models, the order in which a job passes through the machines, known as the processing route, is f xed for each job. For convenience, let us assume that any two consecutive operations of a job are processed on different machines. Other-

wise,these two operations can be comined into a single operation. Two typical models of interest in this context are f ow shop and job shop.

The job shop model is one of the most general models in scheduling theory. In a job shop, each job consists of a number of operations to be processed on all or some of the machines, and each job has its own processing routes to follow. Hence to construct a feasible schedule for a job shop, we have to determine, for each machine, the order in which the jobs are to be processed. Note that in a job shop, a job may visit a machine more than once, and a job may not visit a machine at all. The f ow shop is a special case of the job shop. In a f ow shop, each job requires processing on every machine only once and the processing route is identical for all jobs. In general, jobs are able to pass each other while they are waiting in queues at the machines for processing provided that all jobs follow the same order. Since, no passing among jobs is allowed, the f ow shop is referred to as a permutation f ow shop, and the schedule is said to be a permutation or no-passing schedule.

In the aforementioned shop models, there are no precedence relationships between jobs prescribing the order in which job processing must be carried out. While the machine sequence (i.e., the processing route) of all jobs is given, the scheduling problem is to f nd the best job processing sequence according to a desired optimality principle.

1.1.3 Scheduling classif cation

The seemingly inf nite number of deterministic machine scheduling problems makes it clear that there is a need for classif cation. In general, deterministic machine scheduling problems may be represented using a three-f eld notation, | | , proposed by Graham et.al [13]. Simply, the three-f eld notation , | | , may be sketched as follows:

- The f rst f eld, , indicates the machine environment. For instance, = F or = J denotes the f ow shop or job shop model respectively. The number of machines, *m*, is either part of the problem instance or equal to a f xed constant. In the latter case, the letter *m* or a positive integer is added after the machine environment, e.g., the two machine job shop model is specif ed by J2.

- The second f eld, , consists of the job characteristics, i.e. the processing restrictions and constraints. In contrast to the f rst f eld, this f eld can be empty, which implies the default of non-preemptive and independent jobs. Examples of possible entries in this f eld are = pmtn, meaning that preemption is allowed (i.e. the processing of any operation may be interrupted and resumed at a later time), and = prec, meaning that there are precedence constraints between the jobs (i.e. the processing of a job cannot start before the completion of another job).

- The third f eld, , specif es the optimality criterion or the objective. An optimality criterion assesses the relative merits or performances of competing feasible schedules. Examples of commonly used criteria including minimizing the makespan C_{max}

(i.e. the maximum completion time of all jobs on all machines) and minimizing the total weighted completion time, ($\sum w_j C_j$, of all jobs.)

Stating each variation of a scheduling problem using the three-f eld notation provides a quick reference point to facilitate comparisons between problems. For instance, the problem of minimizing makespan in a *m*-machine permutation f ow shop is identif ed by the three-tuple Fm|$prmu$|C_{max}, while the problem in a general (without permutation) *m* machine f ow shop is denoted by Fm||C_{max}.

Above we have given a rough description of the classif cation scheme. For further details, the reader is referred to the book by Pinedo [24].

1.2 Flow Shop Scheduling

As described earlier, the f ow shop problem is a special case in job scheduling problems. A f ow shop is often referred to as a mass production shop, or is said to have a continuous manufacturing layout. The shop layout (arrangement of machines, benches, assembly lines, etc.) is designed to facilitate a good product f ow. The process industries (chemical, oil, paint etc.) are good examples of f ow shops. Each product, though variable in material specif cations, uses the same f ow pattern through the shop. Production is set at a given rate and the products are generally manufactured in bulk. This study intends to solve two different kinds of f ow shop scheduling problems with minimizing the makespan as a common objective. One of the problem deals with the modif ed f ow shop problem

with release times and due dates. The other problem is a classical f ow shop scheduling problem.

1.2.1 Approaches to solve f ow shop scheduling problem

It is theoretically possible to enumerate all n! possible sequences, by which n jobs might be processed, calculate the corresponding objective function value for each sequence and select the sequence which optimizes the objective function. This total enumeration approach works well for very small size problems. However, as the number of jobs increases, the number of sequences that need to be investigated rapidly grows beyond the bounds of practicality for even today's high speed computers . A complete enumeration of a problem involving only 10 jobs requires investigating 3,628,800 different sequences.

A problem that can be solved in a polynomial number of steps of the input size is said to be P-complete. Other problems fall into a class, known as NP-complete, which are known to be unsolvlabe till to-date in a polynomial time. Garey[9] shows that the f ow shop sequencing problem for large instances is NP-complete. This f nding is important as it indicates that there is no known solution algorithm that can solve the large sized problems in a polynomial number of steps. It is this realization that has led much of the effort to develop heuristics that are likely to produce good but not necessarily optimal solutions.

The literature provides a variety of articles on exact and heuristic approaches developed to solve the f ow shop problems. The exact algorithms such as branch and bound technique

are capable of determining optimal solutions. However, this method still require an inordinate amount of computational effort to solve realistic size problems. Heuristic approach, on the other hand determines a "good" but not necessarily optimal solution. This study focuses on the heuristic approach for solving f ow shop problems.

1.3 Overview of the thesis

The rest of the thesis consists of f ve chapters. For the sake of enhancing the readability of this thesis, Chapters 2 to 4, the main body if the thesis is written so as to be coherent and fully self-explained in the sense that all formal concepts and arguments needed to analyze the problems in each chapter are explained in detail. This implies that some of the chapters may contain redundant material. This chapter provides an introduction that presents suff ciently relevant information to establish a problem context and solution approach.

Chapter 2 is devoted to the similar work done on the f ow shop scheduling problem of n jobs and m machines. The chapter also highlights the work done using the branch and bound technique and also several heuristic approaches. The intent is to use a memory based technique called tabu search. This study tries to pen down as much related work done to solve the focused f ow shop problem using this meta-heuristic approach.

Chapter 3 and 4 are dedicated to provide heuristic solutions to f ow shop scheduling problem and also def ne the problem statement for each case. We provide a new heuristic to solve the f ow shop problem with release times and due dates. Furthermore, we

present a fast approximation algorithm based on tabu search to solve the classical f ow shop problem.

Chapter 5 presents the results of the implementation of the proposed tabu search approache on a set of benchmark problems. Comparisons in quality of solution and computational time are made with the best known results on the classical f ow shop problem to prove the superiority of the approach.

Finally, Chapter 6 draws out the overall implications and advantages of the research.

CHAPTER II

LITERATURE REVIEW

2.1 Introduction

In the current competitive business environment, effective scheduling has become a necessity for survival in manufacturing and service industries. Companies have to meet due dates committed to the customers. They also have to schedule activities to use the resources in an eff cient manner. There is a wealth of literature dealing with job scheduling problems much of which specif cally treats the f ow shop scheduling problems which is the object of this study.

The discussion which follows is not intended to be an exhaustive review of the literature but will instead cite typical examples and discussion about their heuristics which lay the groundwork for the proposed algorithms in this study.

2.2 Exact algorithms

As discussed before, sequencing methods in the literature can be broadly categorized into two types of approaches namely exact and heuristic. Exact solution approach guarantees to obtain the optimum sequence, whereas heuristic approaches mostly obtain near-optimal sequences.

If we had zero idle time, the schedule would be optimal. However, in a flow shop it is impossible to have zero idle time. Since the first job scheduled on machine 2 cannot start until it is completed on machine 1, machine 2 must be idle during this time. Similarly, machine 1 must be idle while the last job is in process on machine 2. Johnson [16] came up with a procedure where he proposed that the idle time on machine 2 must be as long as the shortest processing time on machine 1. Similarly, the unavoidable idle time on machine 1 must be as long as the shortest processing time on machine 2. This leads to a better bound on makespan. Johnson's algorithm is proven to give an optimal solution for the 2 machine case.

The steps in Johnson's algorithm are as follows: Find the shortest processing time among all jobs on both machines. The job with which this time is associated is scheduled last in the sequence, if the shortest time occurs on the second machine. The remaining jobs are then searched for the next shortest processing time, that job is scheduled accordingly and the process is repeated until all jobs are scheduled. This simple algorithm can be extended to optimize *n*-job, 3-machine flow shop problems under certain restrictive conditions. Johnson extended his algorithm to cases where the second machine was dominated by either the first or the third. He then applied the two machine procedure to artificial times created by adding the processing times for each job on the first two (machines 1 and 2) and last two (machines 2 and 3) machines. Attempts to extend Johnson's algorithm to optimizing schedules for more than three machines have not been successful.

Lomnicki [33] proposed a branch and bound technique to find the optimum permutation of jobs. Vaessens [31] solved the classical flow shop problems optimally on a number of benchmarks maintained by Taillard [30] by using branch and bound method method.

2.3 Heuristic methods

This section discusses some heuristic methods from literature which provides insight to solve the problem. The heuristics mentioned here have a common objective to minimize the makespan.

Several heuristic methods were modeled on the application of Johnson's algorithm. Campbell, Dudek and Smith [5] proposed a method known as the CDS algorithm which is a generalization of Johnson's algorithm. This algorithm uses Johnson's rule in a heuristic fashion. The CDS algorithm creates *m-1* schedules from which the best can be chosen. The algorithm generates a set of *m-1*, two machine problems from the original *m* machine algorithm, each of which is then solved using Johnson's two-machine algorithm. The best of the *m-1* solutions becomes the heuristic solution to the m-machine problem.

Dannenbring [7] developed a method called rapid access procedure which constructs an artificial two machine problem with the processing times determined from a weighting scheme. Palmer [22] developed the slope order heuristic, referred to as a quick method of obtaining a near-optimum solution. It is based on the concept that jobs placed early in the sequence should have processing times that tend to increase from one machine to the next machine as the jobs progress through the flow shop, while jobs assigned to late

positions should have decreasing processing times as they move from one machine to the next. Gupta [15] presented an alternative approach for calculating the slope index. The Nawaz [20] algorithm (NEH) is probably the most well-known constructive method used in permutational f ow-shop problems. In NEH, the basic idea is to f rst optimally schedule the two longest processing jobs using Johnson's rule. Then, place the remaining jobs in decreasing order of total processing times, one by one, in one of the slots between already scheduled jobs such that the total makespan is minimized at each step. Rajendran & Chaudhuri [26] further modif ed NEH algorithm by introducing a weighted-sum measure for prioritizing the jobs.

Pour[25] developed an algorithm to minimize maximum f ow time. The algorithm is based on an approach which relies on exchanging one job with another through the sequence until a new combination of jobs is found with respect to a given measure of performance. Travelling Salesman Problem (TSP) requires the salesman to visit the number of cities in his line of work by spending as little time as possible and thus looks for the shortest route to visit all the cities and come back home. Similarly, all jobs have to visit all the machines in the shortest time. Researchers such as Marino, et.al [32] used this analogy to propose an algorithm known as SPIRIT (sequencing problem involving a resolution by integrated taboo (tabu) search techniques) for solving the f ow shop problem to minimize makespan. Sarin and Lefoka [27], and Moccellin [19] have used idle times for the development of constructive heuristics with the objective of minimizing the total

time to complete the schedule (makespan). Taillard [29] studied constructive methods and found that NEH out performs all the other heuristics.

2.3.1 Tabu search

Tabu search (TS) dates back to the 1960's and 1970's and was originally proposed by Glover [10]. The majority of the applications of TS started in late 1980's. Tabu search has been applied to transportation problems such as traveling salesman and vehicle routing, layout and circuit design problems such as quadratic assignment and electronic circuit design, neural networks (non-convex optimization and learning in an associate memory), telecommunications problem such as bandwidth packing, path assignment and, scheduling problems, etc. One of the common applications of TS is production scheduling.

Tabu search works by choosing an initial sequence as a current solution and then applies a move mechanism to search the neighborhood of the current solution to select the most appropiate one. It forces the move, causing the selected solution be forbidden (tabu) for a certain number of iterations and then changes the current solution into the selected one. Despite the simplicity of this approach, it still remains an art in def ning neighborhood, searching among neighbors, def ning forbidden moves, setting tabu list length, etc. Different implementations of these elements result in different tabu search-based algorithms. Basic ideas and advanced topics of TS have been widely discussed in Glover [10]. Widmer and Hertz [32] solved a *n* job, *m* machine f ow shop problem for makespan crite-

rion. Taillard [29] applied improvemnts in his tabu search approach to the same problem resulting in faster calculations.

In local search algorithms the quality of the f nal solution typically increases if a large neighborhood is def ned. However, the larger the neighborhood the longer it takes to search the neighborhood at each iteration. It is therefore crucial to identify a good neighborhood. There are certain block properties used in TS which help identifying good neighbors. Nowicki and Smutnicki [21] used these block properties to explore different sequences in f ow shop scheduling. These properties ensure that a considerable number of moves cannot be effective and therefore should be ignored. This results in saving computation time as well as enhances the quality of solutions. Grabowski and Wodecki [12] presents some new properties associated with the blocks to make the tabu search algorithm faster in computational time. Ben-Daya and Al-Fawzan [2] proposed simple techniques for generating neighborhoods of a given sequence and also a mechanism of intensif cation and diversif cation that has not been considered before. Solimanpur, Vrat and Shankar [28] propose a neuro-dynamical tabu search. Unlike the classical tabu list mechanisms where a move is either tabu or not tabu, their approach shows that the tabu effect declines over time.

We study these neighborhood identifying block properties and propose new neighborhood def nitions to enhance the computational speed and quality with different modif cations of several aspects of classical tabu search method.

CHAPTER III

HEURISTIC APPROACH TO SOLVE THE MODIFIED FLOW SHOP PROBLEM

3.1 Problem statement

We consider the f ow shop problem that consists of *n* jobs, to be processed on *m* different machines. The problem differs from the classical f ow shop in the sense that the jobs have release times and due dates. The following are underlying assumptions to the problem:

- All jobs will follow the same sequence of machines.
- Every machine can process only one job at a time.
- Once a job is started on any machine, it has to be completed without breaking (non-preemption).
- All machines are available at time zero.
- Job *i* is ready for processing at or after its release time, and must be completed by its due date.

The objective is to f nd a sequence of jobs to be processsed such that the makespan is minimized.

We use the following notation:

- p_{ij} presents the processing time of job *i* on machine j
- d_i presents the due date of job *i*
- r_i presents the release time of job *i*

3.2 Motivation

Much research effort has been devoted to the f ow shop problem over the past three decades. The problem has held great interest from a theoretical standpoint because many of the factors which affect the pure f ow shop model are common to other scheduling models that have had more practical applicability. In other words, although a f ow shop model is a simple model, it has found a number of applications. With the exception of some industries like chemical and oil industries, there are few instances of a pure f ow shop to be found. Thus, the primary benef t to be derived from f ow shop research was the insight and understanding gained which could then be transferred to other scheduling problems of a more practical nature. However, the theoretical importance of f ow shop problem has recently gained practical/industrial importance which has indeed extended the ongoing research on this problem.

The ship building industry is an excellent example of the application of the problem. The ship building industry is facing a major challenge to reduce construction lead-times while simultaneously reducing operating costs and increasing f exibility [14]. Current ship building practice relies upon outdated planning, scheduling and coordination techniques across all levels of the ship building enterprise. Management of the panel shop in a ship-yard is a complex process. The panel shop is considered the bottleneck of the shipyard since each panel for every ship must be processed through the shop. For the panel shop problem, the objective is to f nd the best sequence in which to produce the panels, which is subject to material availability and downstream process due dates. “Best” is evaluated

in terms of the throughput of the shop; i.e. get as many panels out of the shop as possible serving the broad objective of minimizing the total completion time of ship building. The panel shop can be viewed as a f ow shop problem where all jobs follow the same sequence of machines and where each job has a due date. Generating a good sequence for the above mentioned problem to reduce the makespan and meet the due dates is the main motivation of this study. The primary objective of this research is the development of heuristic techniques to generate schedules in an effort to minimize makespan that can be applied to problems encompassing some of the realistic conditions found in industrial settings like the ship building industry.

3.3 Solution approach

We propose a heuristic termed EEDERP (Earliest Effective Due dates Effective Release times Processing times) to solve the f ow shop problem with release times and due dates. The core idea of this heuristic method is the identif cation of a bottleneck machine and then according to the priority, scheduling the jobs on it based on effective due dates, effective release times and processing times respectively. The objective of the EEDERP algorithm is to minimize the makespan.

The effective due dates of job *i* on machine *j*, d_{ij}, is the latest point in time by which execution of job *i* on machine *j* must be completed in order for the due dates to be met (d_i for *i*= 1,....,n). In other words, d_{ij} is the latest completion time of job *i* on machine *j*.

The effective release time of job *i* on machine *j*, r_{ij}, is the earliest point in time at which the execution of job *i* on machine *j* can be scheduled. The starting time of job *i* on machine *j* cannot be scheduled prior to having job *i* being processed on machines 1,...,*i*-1.

Effective due dates (d_{ij}) and effective release times (r_{ij}) for all jobs on all machines are calculated by the following equations:

$$d_{ij} = d_i - \sum_{l=j+1}^{m} p_{il} \; and \; r_{ij} = r_i + \sum_{l=1}^{j-1} p_{il} \tag{3.1}$$

According to this algorithm, first we identify a bottleneck machine. The bottleneck machine is the machine which has the job with the maximum processing time on it. Once the bottleneck machine is identified, the Priorities are then assigned to jobs based on their effective due dates. The earlier the effective due date, the higher the priority.

Based on initial observations from literature we found that if a job with smaller processing time is scheduled earlier, the algorithm performs better in terms of makespan. This may affect the overall lateness of jobs to a greater extent. The algorithm schedules the jobs on the bottleneck machine by their earliest effective due date. There can also be a situation when there is a tie in these effective due dates. In such a case, the priority shifts onto effective release times. Moreover, in a situation where the effective release times also has ties, the shortest processing time is used to break the tie. This process continues in the same fashion until all jobs are scheduled. It is also to be mentioned here that we adopted a different way to decide ties. Grouping of effective due dates, effective release times and processing times is done so that the algorithm does not look at exact ties. For instance,The number of groups for the effective due dates can be calculated as $k = \sqrt[3]{n}$. Also the range

is calculated as *r* = largest effective due date -smallest effective due date. By using *r* and *k*, cell width is calculated as $cw = r/k$. In this way all the effective due dates are divided into different groups. For all the jobs with effective due dates in one group are considered as ties. This similar approach is followed for effective release times and processing times.

Figure 3.2 describes the steps of the algorithm in a f ow chart form.

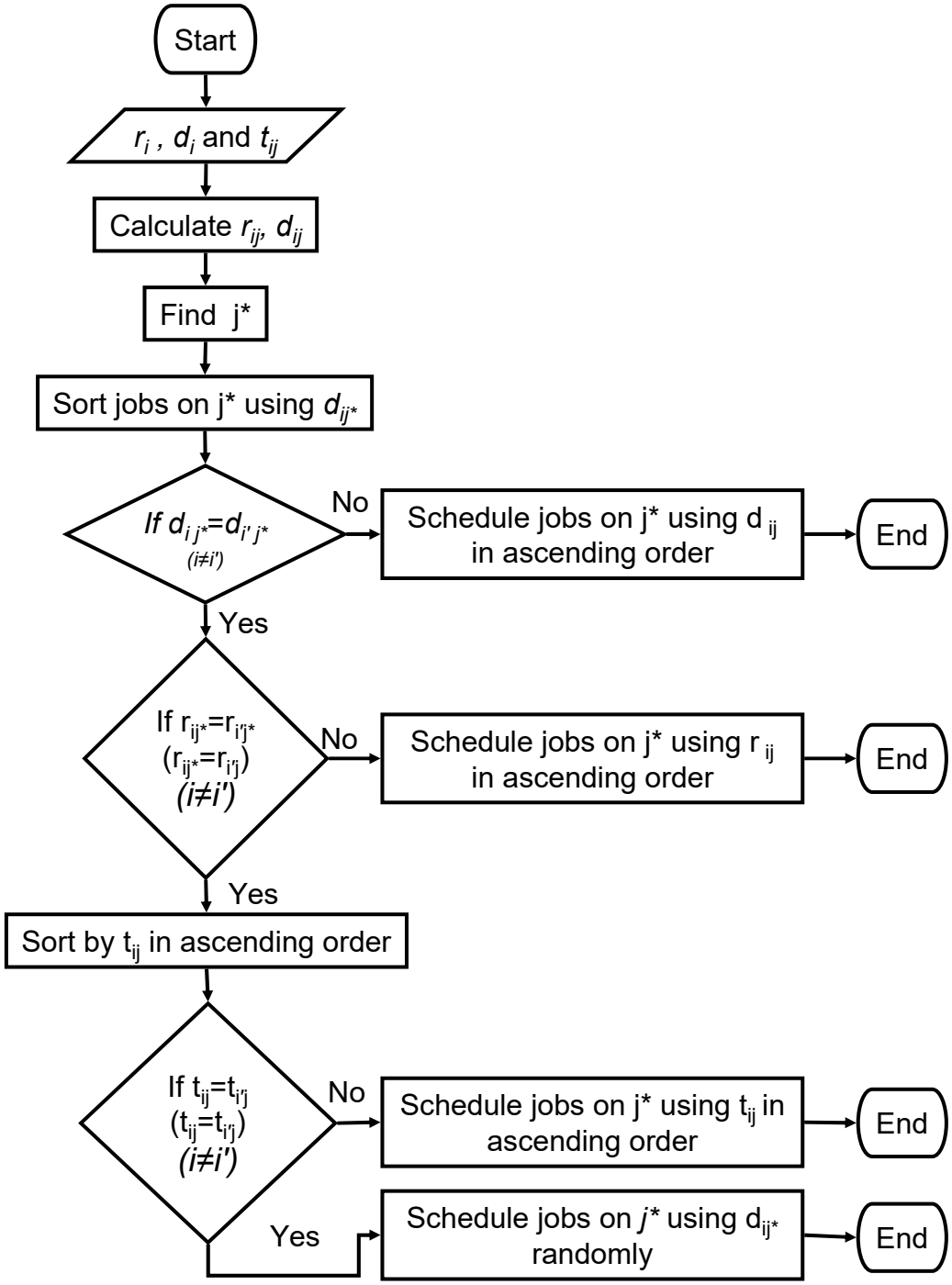

Figure 3.1

Flow chart for EEDERP algorithm

CHAPTER IV

A TABU SEARCH APPROACH FOR CLASSICAL FLOW SHOP PROBLEM

4.1 Introduction

The scheduling problem investigated in this study is called the permutation f ow shop scheduling problem (PSFP) or the classical f ow shop problem and is conventionally designated as n/m/P/C_{max}, where *n* jobs have to be processed on *m* machines in the same order. P indicates that only the permutation schedules are considered, where the order in which each machine possesses the jobs is identical for all machines. Hence, a schedule is uniquely represented by a permutation of jobs. The processing of each job on each machine is an operation, which requires the exclusive use of the machine for an uninterrupted duration which is called the processing time. The objective is to f nd a schedule that minimizes the makespan C_{max}, the time at which the last job is completed on the last machine. As stated earlier, the problem is strongly NP-hard [9]; therefore, exact methods are not computationally practical as the problem size increases. This study solves the f ow shop problem by an iterative improvement method called Tabu Search (TS) method.

4.2 Basic concepts

Figure 4.1 shows an example of schedule π = 4,5,6,1,2,3,8,7 for a problem with n= 8 jobs and m = 6 machines represented by a grid graph, that is taken from Nowicki [21]. In the f gure, the vertical axis corresponds to machines and the horizontal axis to jobs. Each circle represents an operation. The arrows represent precedence relation between operations. A critical path is marked by thin lines.

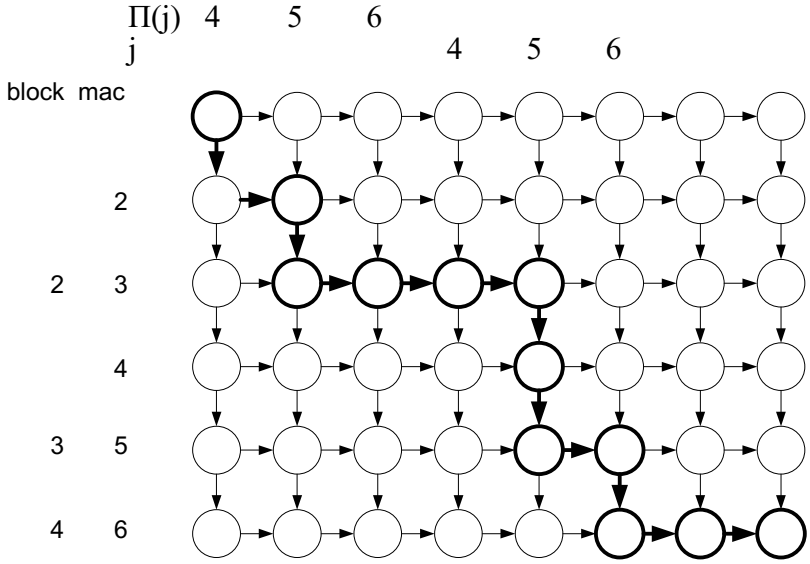

Figure 4.1

Example of critical path

In this example, there are four critical blocks, B_1, B_2, B_3, B_4, that contain more than one job. B_2 on machine 3, for example, consists of four jobs 5,6,1 and 2, and $\hat{B}_2$ consists of jobs 6 and 1. Likewise $\hat{B}_4$ on machine 6 consists of jobs 8 and 7.

A critical path is defined as a sequence of operations starting from the first operation on the first machine, M_1, and ending with the last operation on the last machine, M_m. The starting time of each operation on the path, except for the first one, is equal to the completion time of its preceding operation- that is, there is no idle time along the path. Thus, the length of the critical path is the sum of the processing times of all operations on the path and equals C_{max} of a given schedule. There can be more than one critical path for a given schedule.

The operations on a critical path can be partitioned into subsequences, called critical blocks, according to their associated machines. A critical block consists of maximum consecutive operations on the same machine, or to put it more simply, a subsequence of associated jobs. Most of the following notation has previously been used by Nowicki and Smutnicki [21]. Consider a schedule represented by a permutation π. Let B_1 ,... B_k be a set of all critical blocks that contains more than one job and let m_l be the index of the machine associated with B_l (and the last job of B_{l-1}). Let $\pi(u_l)$ be the first job on B_l (and the last job of B_{l-1}). Then the "inside" of $\hat{B}_l$ is defined as follows:

$$\hat{B}_l = \begin{cases} B_l \setminus \{\pi(u_{l+1})\} & if\ l = 1\ and\ m_l = 1 \\ B_l \setminus \{\pi(u_l)\} & if\ l = k\ and\ m_l = n \\ B_l \setminus \{\pi(u_l), \pi(u_{l+1})\} & otherwise \end{cases} \tag{4.1}$$

A neighborhood N(x) of a point *x* in a search space can be def ned as a set of new points that can be reached from *x* by exactly one transition or move (a single perturbation of *x*). One of the well known transition operators for PFSP is the shift move which takes a job from its current position and re-inserts it in another position. Let *v* = (a,b) be a pair of positions in π. Here, *v* def nes a move that removes the job π(a) from a position *a* and re-inserts it in a position *b*. If $a < b$, the resulting schedule is represented by $\pi_v = \pi(1), \pi(b), \pi(a), \pi(a-1)\pi(a+1), \pi(n)$. A neighborhood N(V,$\pi$) is def ned as the set of all schedules obtained by shift moves in V = $\{(a,b) : b \notin \{a-1, a\}, a, b \in \{1, n\}\}$

Let $W_l(\pi)$ be a set of moves restricted to the inside of B_l, namely $W_l(\pi) = \left\{(a,b) \in V | a, b \in \hat{B}_l\right\}$ and $W(\pi) = \bigcup_{l=1}^{k} W_l(\pi)$, then the so called "block property" is formulated as follows:

Block Property: For any schedule , $\in N(W(\pi), \pi) : C_{max}( ) \geq C_{max}(\pi)$.

According to the block property above, no move in $W(\pi)$ can directly improve schedule π. Therefore, it is reasonable for any computational eff ciency to reduce the size of the neighborhood $N(V, \pi)$ by eliminating moves in $W(\pi)$, and to use a new neighborhood $N(V \setminus W(\pi), \pi)$, which we call here a "critical block neighborhood".

4.3 The tabu search approach

When applied to the permutation f ow shop scheduling, the tabu search algorithm is customarily organized to start at some initial sequence and then move successively among neighboring sequences. At each iteration, a move is made to the best sequence in the

neighborhood of the current sequence which may not be an improved solution. In larger problems, or in those where it is expensive to identify the best of all neighborhood solutions, candidate list procedures are used to limit the alternatives considered. Key variations include strategic oscillation approaches that periodically alternate between constructive and destructive approaches, and that fall outside the customary "local search" template. We do not employ such variants, but refer the reader to [10].

The method forbids (makes tabu) sequences with certain attributes in order to prevent cycling, and guide the search towards unexplored regions of the solution space. This is done using special short and long term memory function, based on dependencies as ref ected in measures of recency and frequency. An important form of short term recency-based memory is embodied in a structure called a tabu list. A tabu list consists of attributes of the latest moves made so that visited sequences are not generated again. The size of the list can be f xed or variable as will discuss later.

In its simplest form, tabu search requires an initial sequence, a mechanism for generating some neighborhood of the current sequence, a tabu list, and a stopping criterion. Additional elements which may be useful are aspiration criterion, intensif cation, and diversif cation scheme. For a more complete description of tabu search, the interested reader is referred to the paper of Glover [10] and book by Glover and Laguna [11].

4.4 The proposed tabu search algorithm

In this section, we describe the implementation of the proposed tabu search approach; namely, *3XTS* for solving the f ow shop sequencing problem where the objective is minimizing makespan. The implementation of each element of *3XTS* is now discussed. This includes:

- Initial solution
- Neighborhood structure
- Selection of best neighbor
- Tabu list and its size
- Stopping criteria

1. Initial solution

   To get an initial starting solution, we considered two heuristics: EEDERP and NEH. EEDERP is the heuristic method developed for solving the f ow shop problem with release times and due dates in chapter 3. In order to use EEDERP for the benchmark problems where there are no release times and due dates, we set the release times to zero and create a common due date, *D*, such that $D \geq C_{max}$. Such a due date can be found by the following equation:

   $$di = (n + m - 1) * maxp_{ij} \tag{4.2}$$

   NEH is the algorithm developed by Nawaz, Enscore and Ham (NEH) [20]. A brief introduction of NEH algorithm is as follows:

- Order the *n* jobs by decreasing total processing time on the machines.
- Consider the first two jobs, and schedule them in order to minimize the partial makespan as if there were only these two jobs.
- For *k* =3 to *n* do:
- The current partial sequence contains *k-1* jobs. Insert the *k*-th job at the position which minimizes makespan among the *k* possible positions available.

As discussed in the literature review, this algorithm is based on the assumption that a job with high total processing time should be given higher priority than a job with lower processing time.

Many heuristic methods have been developed for the flow shop problem, some of which were mentioned in the review, but these two heuristic methods exhibited superiority over others by virtue of their performance in randomly tested problems. NEH algorithm in particular outperformed EEDERP in terms of quality and speed, and therefore will be used as the starting seed for the tabu search algorithm. However, note that EEDERP is tailored for problems with release times and due dates.

2. Neighborhood structure

Given a sequence, *s*, we define N(s) as being the set of all sequences which can be defined from *s* using the following schemes:

- 3-Swap exchange: A neighbor of *s* is obtained by interchanging the jobs in positions *i* and *j* and *k*. The job placed at the *ith* position will be moved to *jth* position. The job placed at the *jth* position will be moved to the *kth* position and the job placed at the *kth* position will be moved to the *ith* position.

- 2-Swap exchange: The 2-swap exchange is done similar to the 3-swap exchange described above. In this case the job in position *i* goes to the position *j* and the job in position *j* moves to position *i*.
- Insertion: Given a sequence, *s*, let *i* and *j* be two positions in the sequence *s*. A neighbor of *s* is obtained by inserting the job in position *i* in position *j*.

Based on the def nitions given above it is easy to show that the size of the 3-exchange neighborhood is n(n-1)(n-2)/3, the size of the 2-exchange neighborhood is n(n-1)/2 and the size of the insertion neighborhood is (n-1)(n-1). The size of the 3-exchange, 2-exchange, and insertion neighborhoods are in the order of O(n3), O(n2), and O(n2) respectively. In general it is more likely to f nd a better neighbor as the size of the neighborhood increases at the expense of more computational time. To search these large neighborhoods (especially 3-exchange) eff ciently several rules are developed.

Let π be the current solution, and let p_1 be a critical path for π on the grid network (see Figure 4.1 as an example). The makespan of the current solution is C_{max}, which is the length of the critical path. Let π' be a neighboring solution to π obtained by performing a 3-exchange (by moving the job in position *i* to position *j*, job in position *j* to position *k* and the job in position *k* to position *i*. Let C_{p1} be the length of the same path for the new sequence. Note that p_1 may not necessarily be the critical

path for π'. Therefore, by def nition $C_{p1} \leq C_{max}$ of π'. C_{p1} can be found by the following equation: $C_{p1} = C_{max}$ for $\pi + \delta$ where

$$\delta = \sum_{l \in m_j} P_{il} - P_{jl} + \sum_{l \in m_k} P_{jl} - P_{kl} + \sum_{l \in m_i} P_{kl} - P_{il} \tag{4.3}$$

m_j = the set of machines that the job in position *j* is on w.r.t the path p_1, m_k = the set of machines that the job in position *k* is on w.r.t the path p_1 and m_i = the set of machines that the job in position *k* is on w.r.t the path p_1.

After combining the above equations we get Cmax of $\pi' - C_{max}$ of $\pi \leq \delta$. This relationship indicates that exchanges for which $\delta \geq 0$ are not worth exploring. We f rst calculate the delta value of a move before we calculate the corresponding makespan. This saves signif cant computational time since makespan calculation is expensive. It has been noted that makespan calculation takes about 80-85 % of the total computational time for a heuristic method that solves f owshop scheduling problems [2]. Note that this is an extension of the block property given by [21]. Nowicki [21] shows that exchanging jobs within a block will not lead to a better solution. Using our equation we can show that δ = 0 for those exchanges within a block. However, we can also eliminate other exchanges that result in $\delta \geq 0$.

Limited number of neighbors are searched randomly,instead of searching the whole neighborhood to further decrease computational time. As soon as a better neighbor is found, the move is made to that neighbor. If a better neighbor is not found af-

ter investigating a predetermined number of neighbors, a move is made to the best neighbor found so far.

3. Selecting the best neighbor in the candidate list

   The objective function is the makespan. Thus, we def ne "best" by referencing the objective function and the current tabu conditions. The best neighbor on the candidate list is the sequence that yields the smallest makespan without creating a tabu move. We allow our candidate list size to be even smaller than indicated above by not undertaking a full search of the list, but by a predef ned maximum number of neighbors to be searched, and accepting the f rst non-tabu move that improves the current solution. If there is no candidate move that improves the solution, we examine the neighborhood of the current sequence and accept the neighbor which has the best makespan. This best neighbor sequence then becomes the current sequence and the search starts again from this sequence. The idea behind this logic is to not to get trapped in the local optima but to strive to search for the global optima.

4. Tabu list

   The size of the tabu list is a very important parameter of a tabu search algorithm. The tabu list can be either f xed or variable. The size of the tabu list may be f xed to some value, for example 7. This means that the tabu list contains seven prohibited moves. Taillard [29] has shown that for an insertion method, the size of tabu list, if seven, yields better results. Since we employ 3-exchange, 2-exchange and insertion

together, we keep three different tabu lists for these exchanges with different sizes. It is diff cult to f nd eff cient values of tabu list sizes, mostly tedious trial and error process are employed. *3XTS* tunes the value of tabu list size for all the three tabu lists as a function of problem size. The details of which will be talked about in the results section.

5. Stopping criteria

   We start the *3XTS* algorithm with the 3-exchange method, followed by the two exchanges and then f nally insertion. The starting sequence for 3-exchange is provided by the NEH algorithm which is also the best initial sequence. The best keeps updating whenever a new sequence with a better makespan is found. The starting sequence for the 2-exchange and insertion is the best sequence. *3XTS* has different stopping criterions for the algorithm to move out of 3-exchange, 2-exchange, insertion, and to terminate. The maximum number of iterations for each procedure is def ned initially, and the number of bad iterations (bad moves) is compared to the maximum iterations to get out of the procedure for each case. The whole algorithm repeats execution for a def ned number and is terminated. The best sequence and its corresponding makespan after all the runs is the f nal solution.

This completes the description of our implementation of the tabu search algorithm, *3XTS*. Figure 4.2, Figure 4.3, Figure 4.4 and Figure 4.5 explains in detail these implementation steps.

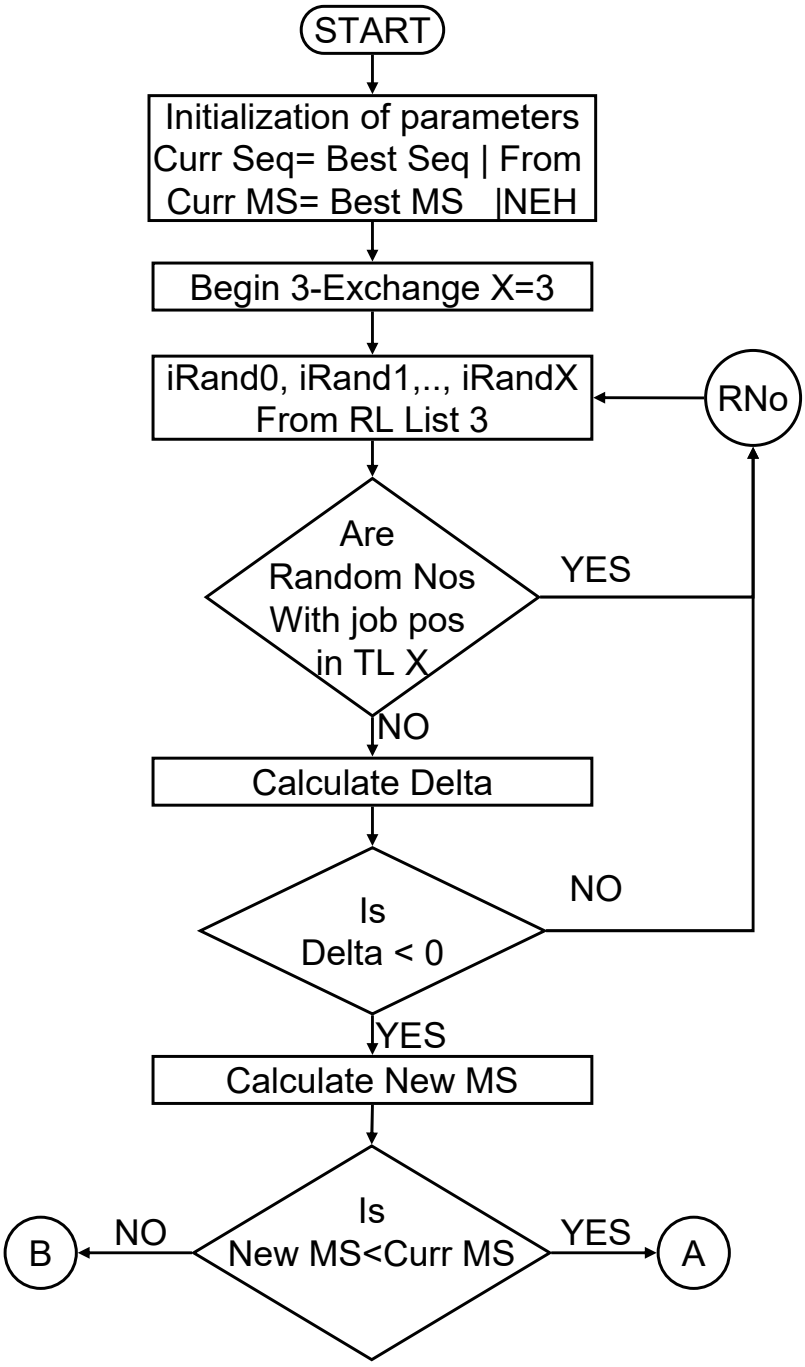

Figure 4.2

Flow chart for 3XTS algorithm

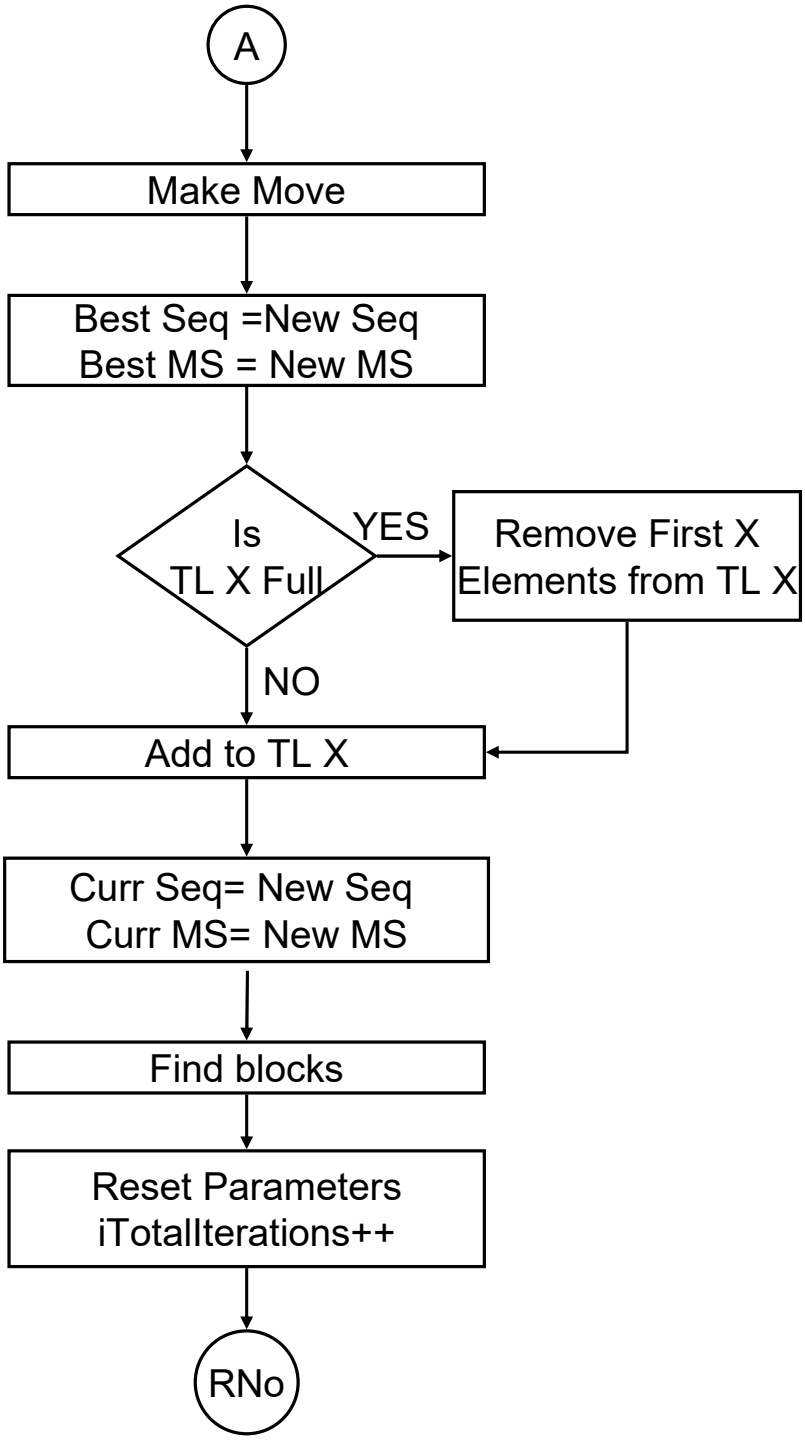

Figure 4.3

Flow chart for 3XTS algorithm (Contd.)

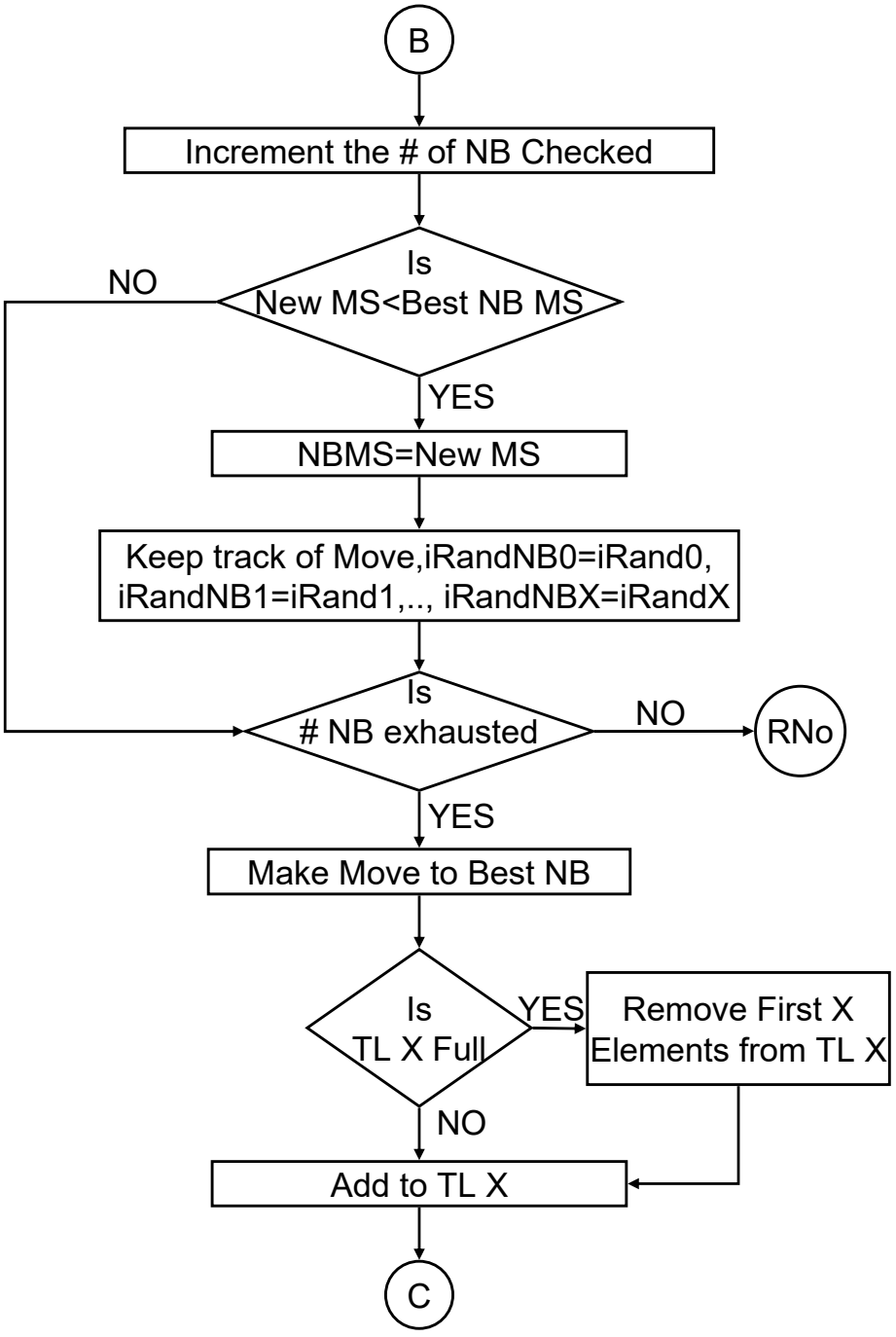

Figure 4.4

Flow chart for 3XTS algorithm (Contd.)

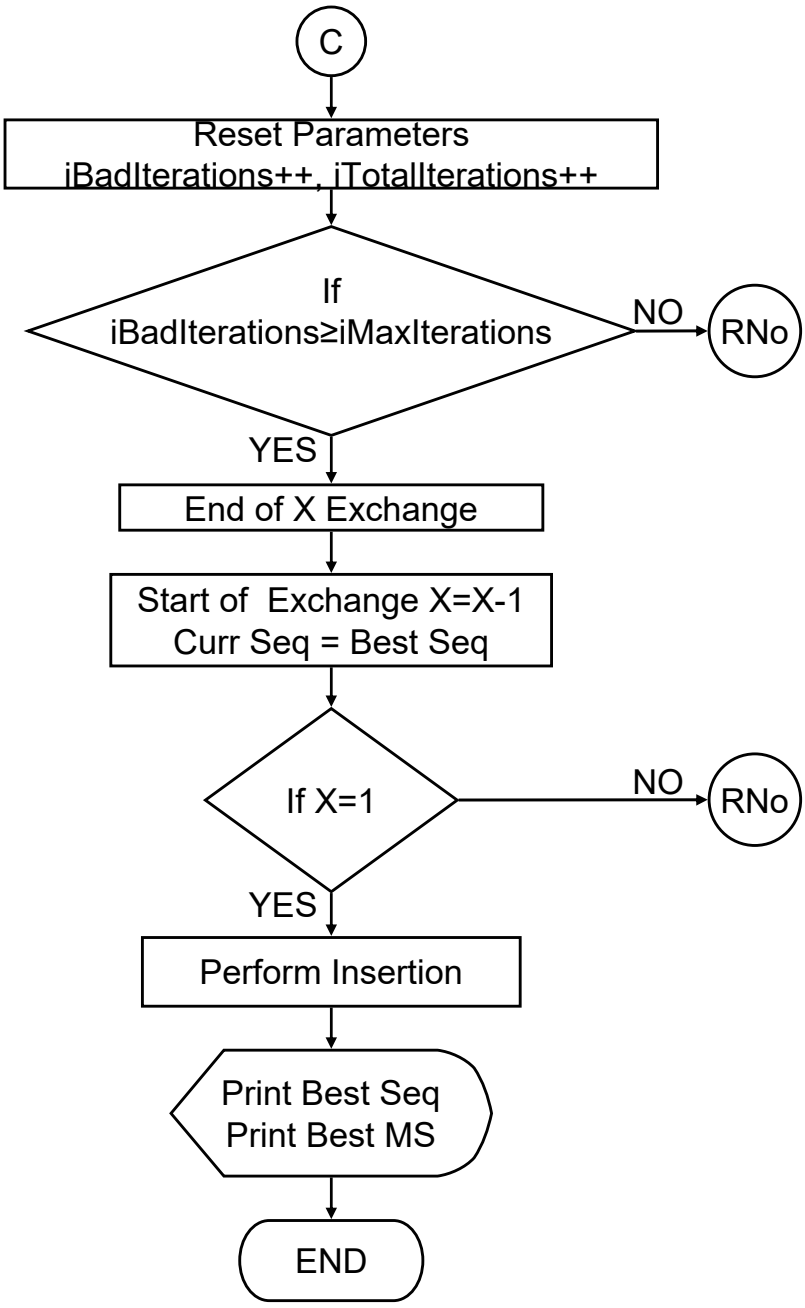

Figure 4.5

Flow chart for 3XTS algorithm (Contd.)

The algorithm starts with the initialization of all the parameters and the initial sequence via NEH algorithm. This initial sequence is the current sequence for the algorithm and initial calculated makespan, its corresponding current makespan. The best sequence and best makespan is also updated to current sequence and makespan respectively. *3XTS* starts with 3-exchange method. In the 3-exchange method, three random numbers *iRand0, iRand1* and *iRand2* are picked up randomly from the random list 3. The algorithm then checks in tabu list3 for these random numbers and the positions of jobs associated with them. If the tabu list contains these elements, the algorithm goes back again to generate a new set of random numbers. If the tabu list is void of these elements, delta (δ) is calculated and checked for its negativeness. In case of a positive or a zero δ, the algorithm goes back to the initial state where three new random numbers are generated. A negative δ allows the algorithm to calculate a new makespan (New MS), and then this New MS is compared to the current makespan (Curr MS).

If New MS $<$ Curr MS, then a move to a new neighborhood is made and the new makespan is updated. The best sequence and best makespan are also updated simultaneously. Then, the tabu list 3 is checked for its size. If it is full, the f rst three elements added to it are removed and three new elements are added. The current sequence and current makespan are then updated with the new sequence and its corresponding makespan. The blocks are then calculated based on the current sequence and all the parameters are reset with the number of total iterations increased. The algorithm goes back to the beginning where a new set of random numbers are generated.

If New MS $<$ Curr MS, the number of entries checked for in a neighborhood is increased. The new makespan is then compared with the best neighborhood makespan. If the New Ms is better ,the track of random numbers leading to the best neighborhood sequence is done (move is tracked). The number of neighborhood entries are checked for additional possibilities. If they are not, the algorithm goes back to generate a new set of random numbers and starts from the very beginning. If the new makespan is less than the best neighborhood makespan, the number of entries for neighborhood entries is again checked for more possibilities. Again, exhausting these entries leads the algorithm back to the beginning where new random numbers are generated. If the number of neighborhood entries are not exhausted, a move is made to the best neighborhood with the tracked random numbers. The tabu list3 is again checked for size. If it is full, the first three elements are added and new elements are added. The parameters are reset, and number of bad iterations and total iterations are increased. To stop the 3-exchange method, number of bad iterations are compared with maximum iterations. If ibaditerations $\geq$ imaxiterations, 3-exchange is stopped and 2-exchange method starts with the best sequence as the current sequence. If the stopping condition for 3-exchange is not satisfed, the algorithm goes back to the beginning.

The 2-exchange and insertion methods are very similar but vary in terms of random lists and assosciated random numbers. Both have random lists of different neighborhood size and out of the lists two random numbers are chosen to generate a new sequence.

The tabu list also adds and removes two elements at a time. The rest of the steps in implementation are quite identical to the 3-exchange as shown in the f ow chart above.

CHAPTER V

COMPUTATION RESULTS

5.1 Computational results for the proposed tabu search algorithm

The TS-based method proposed by Ben-Daya and Al-Fawzan [2] used the test problems proposed by Talliard [30] and reprted that the BF-TS method performs better than the other best known TS-based methods. Solimanpur, Vrat and Shankar [28] further compared their *EXTS* algorithm with BF-TS and have shown *EXTS* to outperform the results in terms of solution quality and computational time. Consequently, the *EXTS* algorithm is more effective than all the known TS-methods mentioned in the literature above, and therefore, we compare our results with those obtained through *EXTS* algorithm.

The results related to computational time of *EXTS* algorithm have been obtained through a 550 MHz Pentium-3 PC. To compare the results fairly with the *EXTS*, *3XTS* algorithm has been coded in Visual C and run on 550 Mhz Pentium-3 PC. We have selected the 23 test problems attempted by Solimanpur, Vrat and Shankar [28] for comparison.

The proposed *3XTS* method contains nine parameters, viz. iNB1, iNB2, iNB3, min1, min2, min3, TS1, TS2, and TS3. These parameters affect the performance of the *3XTS* algorithm. Based on experimental observations, we have tuned the values of these param-

eters and derived the rules given in Table 5.1. These rules are used to set the parameters in different problems.

Table 5.1

Rules to set values of parameters

Size	3-exchange			2-exchange			Insertion		
	iNB1	min1	TS1	iNB2	min2	TS2	iNB3	min3	TS3
n=5,10 m =20	2 * n	1000	20* 3	3 * n	10	10 * 2	n	10	7 * 1
n > 10 m =20	2 * n	1000	20* 3	n	10	10 * 2	n	10	7 * 1
n=5, 10 m =50	4 * n	5000	20* 3	n	10	10 * 2	n	10	7 * 1
n >10 m =50	3 * n	5000	20* 3	n	10	10 * 2	n	10	7 * 1
n=5, 10 m =100	4 * n	5000	20* 3	n	1000	10 * 2	n	1000	6 * 1
n >10 m =100	6 * n	5000	55* 3	4 * n	1000	40 * 2	n	100	8 * 1
n=5, 10 m =200	3 * n	10000	20* 3	n	5000	10 * 2	n	1000	7 * 1
n >10 m =200	4 * n	10000	50* 3	2 * n	5000	32 * 2	n	1000	7 * 1

Table 5.2 shows the makespan and the computation time of *3XTS* and *EXTS* methods for the tested problems. As seen in Table 5.2, in 7 out of the 23 problems (problems 8, 9, 17, 20, 21, 22 and 23), the proposed *3XTS* method obtains a lesser makespan then the *EXTS* algorithm. The opposite is true in two problems (problems 6 and 12).

The comparison in Table 5.2 shows the results obtained for a initial set of Talliard data.

The last column of Table 5.2 shows the percentage reduction in the computation time (PRCT%) of *3XTS* compared to the *EXTS* algorithm. As shown in this column, the *3XTS* algorithm has signif cantly reduced the computation time for all but two problems (problems 17 and 22). In problem 17, the *3XTS* algorithm increased the computation time

Table 5.2

Results comparison table

No.	Problem Size	EXTS		3XTS		PRCT()
	N * M	Cmax	Time [a] (s)	Cmax	Time[a] (s)	
1	20 * 5	1278	1.30	1278	0.32	75.38
2	20 * 5	1359	1.50	1359	0.41	72.66
3	20 * 5	1081	2.20	1081	0.72	67.27
4	20 * 5	1293	3.00	1293	1.30	56.67
5	20 * 5	1235	1.50	1235	0.36	76.00
6	20 * 10	**1582**	5.20	1583	2.13	59.03
7	20 * 10	1659	11.10	1659	4.61	58.46
8	20 * 10	1499	5.20	**1496**	3.87	25.57
9	20 * 10	1378	8.50	**1377**	3.92	1.17
10	20 * 10	1419	5.40	1419	2.36	5.19
11	20 * 20	2297	22.40	2297	0.89	96.02
12	20 * 20	**2101**	31.50	2103	6.36	79.80
13	20 * 20	2330	19.50	2330	7.62	60.92
14	20 * 20	2229	17.80	2229	51.37	69.83
15	20 * 20	2291	25.20	2291	8.92	64.60
16	50 * 5	2724	0.82	2724	0.56	31.70
17	50 * 10	3034	4.60	**3025**	7.30	**-58.69**
18	50 * 20	3893	575.40	3893	15.45	97.31
19	100 * 5	5493	4.90	5493	3.24	33.87
20	100 * 10	5771	19.30	**5770**	16.32	18.25
21	100 * 20	6326	242.80	**6300**	61.24	74.77
22	200 * 10	10872	154.40	**10869**	221.32	**-43.34**
23	200 * 20	11326	421.50	**11251**	320.67	23.92
Average reduction in computation time						45.49

[a]Computation time is in seconds obtained on a 550 MHz Pentium III PC.

by 58.69% and in problem 22 by 43.34%. However, in both these cases the makespan improved. The last row of Table 5.2 indicates that the *3XTS* algorithm has reduced the computation time by 45.49% on an average.

Another interesting observation in these experiments is the possibility of higher effectiveness and computational effeciency of the proposed *3XTS* algorithm. For example in problem 23 with 200 jobs and 20 machines, the makespan obtained by the *3XTS* method is 11251 and it is 11326 for *EXTS*. There is a 23.92% reduction in computation time compared to *EXTS* algorithm.

As stated, the size of tabu lists is not kept f xed for all the problems in the experiments. Tabu lists have been assigned values based on the rules given in Table 5.1. *3XTS* is sensitive to this parameter and it has been observed that the 3-exchange in particular with large neighborhood size and large tabu list size performs well. One can argue that, a large tabu list size would result in a quick convergence on a local optimum solution, however, an algorithm allowed to search a large neighborhood performs better and faster. It provides more diversif cation for 2-exchange and insertion methods which results in a better solution. We also observed that insertion or 2-exchange on their own, do not perform as well as the combination of the three methods used together. The use of block properties and δ, makes the algorithm faster in producing results.

CHAPTER VI

CONCLUSIONS AND IMPLICATIONS

6.1 Discussion and conclusion

In this thesis, a heuristic method to solve the f ow shop problem with release times and due dates, is developed. The motivation for the problem came from the ship building industry. The panel shop in the shipyard is modeled as a f ow shop problem. Each job has its own release times and due dates. Much of the literature is focused on different f ow shop problems without due dates. One of the objective of this study is to provide a heuristic method which takes release times as well as due dates into consideration. The objective is minimizing the makespan. The algorithm *EEDERP* is based on f nding the bottleneck machine and scheduling the jobs on it based on priority. In case of ties, the priority is given to effective due dates, then effective release times, and f nally processing times.

Another algorithm *3XTS*, is developed to solve the classical f ow shop problem also called as the permutation f ow shop problem. The *3XTS* is based on the tabu search method. *3XTS* is an improvement metho and gets an initial permutation from a constructive method, - the NEH algorithm. *3XTS* exploits the “block properties” for a faster and better neighborhood search. The proposed *3XTS* algorithm is different from the other TS-

based methods in the sense that it uses 3-exchange, 2-exchange and insertion collectively, which makes the algorithm untrapped in local optimal and gives better results. Based on experiments, some rules are presented to set the values of control parameters. *3XTS* is used on 23 problems adopted from Taillard [30] and compared with the *EXTS* method proposed by Solimanpur, Vrat and Shankar [28]. The experiments observations verif ed the effectiveness and effeciency of *3XTS* over the *EXTS* method. We can summarize the following advantages for the proposed *3XTS* algorithm:

- The *3XTS* integrates 3-exchange, 2-exchange and insertion together, thus diversifying the neighborhood search to yield better results. It also uses block properties and a paramter, δ, which minimizes computation time.
- Rules are provided to set the paramters for the *3XTS* algorithm in different problems. These rules are based on a number of experimental tests.
- *3XTS* introduces a new mechansim in the application of TS- method that has the potential to reduce the computational time and improve the makespan. *3XTS* can be applied to other areas of combinatorial optimization with appropiate modif cations.

6.2 Future direction

This algorithm can be further modif ed to incorporate *X*-exchanges where X=*1,2,...n*. Also, appropriate changes depending on the problem can be made in the algorithm for its application in other optimization problems.

REFERENCES

[1] K. Baker, *Introduction to Sequencing and Scheduling*, John Wiley and Sons Inc., New York, 1974.

[2] M. Ben-Daya and M. Al-Fawzan, "A tabu search approach for the f ow shop scheduling problem," *European Journal of Operational Research*, vol. 109, 1998, pp. 88–95.

[3] J. Blazewicz, K. Ecker, G. Schmidt, and J. Weglarz, *Scheduling computer and manufacturing processes*, 2nd edition, Springer-Verlag, Berlin, Germany, 2001.

[4] P. Brucker, *Scheduling algorithms*, 3rd edition, Springer-Verlag, Berlin, Germany, 2001.

[5] H. G. Campbell, R. A. Dudek, and M. L. Smith, "A Heuristic Algorithm for the n Job, m Machine Sequencing Problem," *Management Science*, vol. 16, no. 10, 1970, pp. 630–637.

[6] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to algorithms*, 2nd edition, The MIT Press, Cambridge, MA, 2001.

[7] D. Dannenbring, "An evaluation of f ow-shop sequencing heuristics," *Management Science*, vol. 23, 1977, pp. 1174–1182.

[8] T. Ganesharajah, N. Hall, and C. Sriskandarajah, "Design and operational issues in AGV-served manufacturing systems," *Annals of Operations Research*, vol. 76, 1998, pp. 109–154.

[9] M. Garey and D. Johnson, *Computers and Intractability*, WH Freeman, San Francisco, 1979.

[10] F. Glover, E.Taillard, and D. de Werra, "A user's guide to tabu search," *ORSA Journal on Computing*, vol. 41, 1989, pp. 3–28.

[11] F. Glover and M. Laguna, *Tabu search*, Kluwer Academic Publishers, Boston, 1997.

[12] J. Grabowski and M. Wodecki, "A very fast Tabu search algorithm for the permutation f ow shop problem with makespan criterion," *Computers & Operations Research*, vol. 31, no. 11, 2004, pp. 1891–1909.

[13] R. Graham, E. Lawler, J. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," *Annals of Discrete Mathematics*, vol. 5, 1979, pp. 287–326.

[14] A. G. Greenwood, T. W. Hill, J. W. Miller, S. Vanguri, B. Eksioglu, P. Jain, and C. T. Walden, "Simulation Optimization Decision Support System for Ship Panel Shop Operations," *To be Published in Winter Simulation Conference Proceedings*, 2005.

[15] J. Gupta, "A functional heuristic algorithm for the f ow-shop scheduling problem," *Operations Research Quarterly*, vol. 22, 1971, pp. 39–47.

[16] S. Johnson, "Optimal Two and Three Stage Production Schedules with Set-Up Times Included," *Naval Research Logistics Quarterly*, vol. 1, no. 1, March 1954, pp. 61–68.

[17] V. Kats and E. Levner, "Cyclic scheduling in a robotic production line," *Journal of Scheduling*, vol. 5, 2002, pp. 23–41.

[18] C. Lee, L. Lei, and M. Pinedo, "Current trends in determinisic scheduling," *Annals of Operations Research*, vol. 70, 1997, pp. 1–41.

[19] J. V. Moccellin, "A New Heuristic Method for the Permutation Flow Shop Sequencing Problem," *Journal of the Operational Research Society*, vol. 46, 1995, pp. 883–886.

[20] M. Nawaz, E. E. E. Jr, and I. Ham, "A Heuristic Algorithm for the m-Machine,n-Job Flow-shop Sequencing Problem," *OMEGA The International Journal of Management Science*, vol. 11, no. 1, 1983, pp. 91–95.

[21] E. Nowicki and C. Smutnicki, "A fast tabu search algorithm for the permutation f ow-shop problem," *European Journal of Operational Research*, vol. 91, 1996, pp. 160–175.

[22] D. Palmer, "Sequencing jobs through a multi-stage process in the minimum total time- A quick method of obtaining a near optimum.," *Operations Research Quarterly*, vol. 16, 1965, pp. 101–107.

[23] C. Papadimitriou, *Computational Complexity*, Addison-Wesley Publishing Company, MA, 1994.

[24] M. Pinedo, *Scheduling: Theory, algorithms, and systems*, 2nd edition, Prentice Hall, Upper Saddle River, NJ, 2002.

[25] H. D. Pour, "A new heuristic for the *n*-job, *m*-machine f ow-shop problem," *Production Planning and Control*, vol. 12, no. 7, 2001, pp. 648–653.

[26] C. Rajendran and D. Chaudhuri, "An eff cient heuristic approach to the scheduling of jobs in a f owshop," *European Journal of Operational Research*, vol. 61, 1991, pp. 318–325.

[27] S. Sarin and M. Lefoka, "Scheduling Heuristic for the n-Job m-Machine Flow Shop," *OMEGA The International Journal of Management Science*, vol. 21, no. 2, 1993, pp. 229–234.

[28] M. Solimanpur, P. Vrat, and R. Shankar, "A neuro-tabu search heuristic for the f ow shop scheduling problem," *Computers & Operations Research*, vol. 31, 2004, pp. 2151–2164.

[29] E. Taillard, "Some effecient heuristic methods for the f ow shop sequencing problem," *European Journal of Operational Research*, vol. 47, 1990, pp. 65–67.

[30] E. Talliard, "http://ina.eivd.ch/Collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html, (current 5 Dec. 2000)," .

[31] R. Vaessens, E. Aarts, and J. Lenstra, "A local search template," *Unpublished Manuscript*, 1995.

[32] M. WIDMER and A. HERTZ, "A new heuristic method for the f ow shop sequencing problem," *European Journal of Operational Research*, vol. 41, 1989, pp. 186–195.

[33] ZA.Lomnicki, "A branch and bound algorithm for the exact solution of the three-machine scheduling problem," *Operations Research Quarterly*, vol. 16, no. 1, 1965, pp. 101–107.